File System Implementation

Disk Structure

partitions: disk can be subdivided into partitions

raw usage: disks/partitions can be used raw (unformatted) or formatted with file system

volume: entry containing FS

— tracks that file system's info is in device directory or volume table of contents

FS diversity: there are general purpose and special purpose FS

File Systems — logical vs. physical

logical: can consist of different physical file systems

placement: file system can be mounted at any place within another file system

mounted local root: bit in i-node of local root in mounted file system identifies this directory as mount point

File Systems — layers

layer 5: applications

layer 4: logical file system

layer 3: file-organization module

layer 2: basic file system

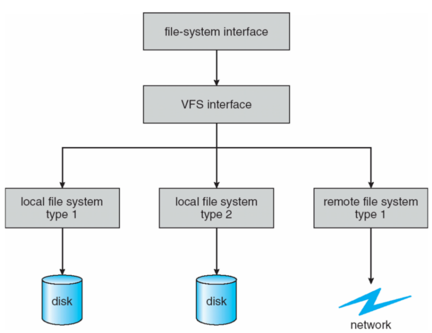
layer 1: I/O control

layer 0: devices

File Systems — virtual

principle: provide object-oriented way of implementing file systems

— same API used for different file system types



Files — implementation

meta data must be tracked:

— which logical block belongs to which file?

— block order?

— which blocks are free for next allocation?

block identification: blocks on disk must be identified by FS (given logical region of file)

$\longrightarrow$ meta data needed in *file allocation table*, *directory* and *inode*

block management: creating/updating files might imply allocating new/modifying old disk blocks

Allocation — policies

preallocation:

— *problem*: need to know maximum file size at creation time

— often difficult to reliably estimate maximum file size

— users tend to overestimate file size to avoid running out of space

dynamic allocation: allocate in pieces as needed

Allocation — fragment size

extremes:

— fragment size = length of file

— fragment size = smallest disk block size (= sector size)

trade-offs:

— *contiguity*: speedup for sequential accesses

— *small fragments*: larger tables needed to manage free storage and file access

— *large fragments*: improve data transfer

— *fixed-size fragments*: simplifies space reallocation

— *variable-size fragments*: minimizes internal fragmentation, can lead to external fragmentation

Allocation — file space

contiguous

chained

indexed:

— fixed block fragments

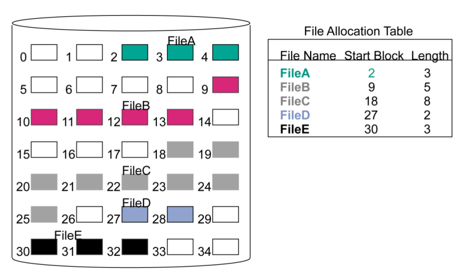
— variable block fragments

characteristic	contiguous	chained	indexed	
preallocation?	necessary	possible	possible	
fixed or variable size fragment?	variable	fixed	fixed	variable
fragment size	large	small	small	medium
allocation frequency	once	low to high	high	low
time to allocate	medium	long	short	medium
file allocation table size	one entry	one entry	large	medium

Allocation — contiguous

principle: array of n contiguous logical blocks reserved per file (to be created)

periodic compaction: overcome external fragmentation



Allocation — chained

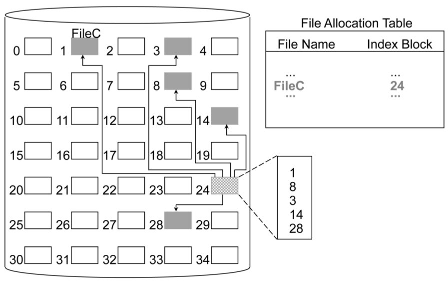
principle: linked list of logical blocks per file

— FAT or directory contains address of first file block

$\longrightarrow$ *no external fragmentation*: any free block can be added to chain

Allocation — indexed

principle: FAT contains one-level index table per file
— *generalization*: n-level index table
— index has one entry for allocated file block
— FAT contains block number for index



Directories — implementation

simple directory (MS-DOS):
— fixed-size entries
— disk addresses + attributes in directory entry

i-node reference directory (UNIX):
— entry refers to i-node containing attributes

Disk Blocks — buffering

buffering: disk blocks buffered in main memory

access: buffer access done via hash table
— blocks with same hash value are chained together

replacement: LRU

management: free buffer is managed via doubly-linked list

File Systems — journaling

principle: record each update to file system as *transaction*
— written to log

committed transaction = written to log
$\rightarrow$ *problem*: file system may not yet be updated

writing transactions from log to FS is asynchronous

modifying FS $\rightarrow$ transaction removed from log

crash of file system $\rightarrow$ remaining transactions in log must still be performed

File Systems — log-structured

principle: use disk as circular buffer
— write all updated (including i-nodes, meta data and data) to end of log

buffering: all writes initially buffered in memory

writing: periodically write within 1 segment (1 MB)

opening: locate i-node, find blocks

clearing: clear all data from other end, no longer used